Introduction to Programming with Jini

Simon Roberts
Sun Microsystems, Inc.

simon.roberts@sun.com
Jini Principles

- Spontaneous and self-healing
- Network reliability
- Zero-administration
- Protocol agnostic
Jini Programming Elements

- Leases
- Distributed events
- Lookup services
- (Transactions)
- (Configuration)
- (Exporter)
- (Security)
Marshalled Objects

- Serialized state
- Reference to behavior
- Avoids need to pre-install classes
  ➢ Critical to distributed “federations”
Moving Objects

- Proxies for services
- Service user interface
  - Appropriate to specific client
- Intelligent proxies (HOPP)
- “Auto-updates”
- Execute client code on server *e.g.*
  - Searches
  - Computationally demanding algorithms
Lease Concepts

- Supports DGC of resources
  - Not just memory
  - Not required if no resource held by server
- Renew lease to indicate
  - Continued client interest
  - Continued client life
- Cancel lease to indicate end of use
Lease Concepts

- Leases are not guarantees
  - Network reliability prevents this
  - Leases are for the provider’s benefit
  - Clients notice failure when using the leased resource
- Long leases reduce network load
- Short leases improve responsiveness
Distributed Event Concepts

- Small
- Leased
- Listener suitable for remote use
- Suitable for unreliable networks
  - Loss, duplication, reordering
Lookup & Discovery Concepts

- Lookup services *a.k.a.* registrars
- Registrars are repositories of service proxies
- Service proxies are the downloadable part of a service
  - State and *behavior*
  - Might be an RMI stub
  - Might be an entire program
  - Must be *Serializable*
Lookup & Discovery Concepts

- Clients and services “discover” registrars
  - Unicast and multicast discovery
- Clients obtain service proxies from lookup services
  - By polling
  - By notification (events)
- Service proxies are described by interface
  - “Capabilities” c.f. Yellow Pages
Lookup & Discovery Concepts (Continued)

- Service providers must maintain lease
  - Otherwise proxy is removed from registrar
- Removal of a proxy can trigger events to notify clients
- Proxies are registered along with “attributes”
  - Clients can search on attributes
  - Attributes are serializable objects
  - Attributes are state and behavior (e.g. GUIs :)
The Lease Interface

- `net.jini.core.lease.Lease`
- `void renew(long duration)`
- `void cancel()`
- `long getExpiration()`
- `ANY and FOREVER`
The LeaseRenewalManager

- com.sun.jini.lease.LeaseRenewalManager
- void renewFor(Lease, long duration, LeaseListener)
- void renewUntil
- void cancel(Lease)
- void remove(Lease)
Implementing Leases

- Generally responsibility of service provider
  - Typically create your own
- Landlord framework reduces number of remote objects, aggregating many leases on a single remote object
- Expiry can be more or less enthusiastic according to your needs
Distributed Events

- Support unreliable networks
  - Event is (should be) small
  - Registration is leased
  - Event carries sequence number

- All events handled by single listener type
  - net.jini.core.event.RemoteEventListener

- Events carry an ID identifying their registration
  - Event ID is *not* an “event reason code”
Event Registration

- Provide:
  - Listener proxy
  - Lease request
  - Marshalled object

- Registration method not mandated
  - Often
    ```
    EventRegistration notify(RemoteEventListener, long, Serializable)
    ```
Event Registration

- Response typically returns an `EventRegistration` object:
  - Event ID
  - Source
  - Lease
  - Initial sequence number
Event Delivery

RemoteEvent object

- Event ID
- MarshalledObject
- Sequence number
- Source
Marshalled Object

- Marshalled object provided during registration
  - Aka “handback object”
  - Returned unchanged to listener

- State and behavior
  - Might be a helper for a generalized listener
  - E.g. filter for frequent events to minimize load on small system
Third-party Event Handling

- All events handled by the same listener interface
- Allows generic handlers (like Unix pipes)
- Event pre-processing can also be achieved by smart listener proxy
  - Not simply a proxy, but computes in event source
- Event mailbox, useful for intermittent connection
Finding a Registrar

- **Unicast**
  - LookupLocator
  - LookupLocatorDiscovery

- **Multicast**
  - LookupDiscovery
  - LookupDiscoveryManager
LookupDiscoveryManager

- Constructor arguments
  - `java.lang.String[] groups`,
  - `LookupLocator[] locators`,
  - `DiscoveryListener listener`

- `DiscoveryListener` declares
  - `discovered(DiscoveryEvent)`
  - `discarded(DiscoveryEvent)`

- `DiscoveryEvent.getRegistrars()`
ServiceRegistrar

- Three primary operations:
  - register(ServiceItem, long)
  - lookup(ServiceTemplate)
  - notify(ServiceTemplate, int, RemoteEventListener, MarshalledObject, long)
Registration

- **register** method of `ServiceRegistrar`
  - Takes `ServiceItem`
  - Returns `ServiceRegistration`
ServiceItem

- Service proxy described as an Object in the ServiceItem
- ServiceID
- Object
- Entry []
ServiceRegistration

- Gives access to service ID, lease, and attributes
- Universally unique service ID
Service Provider Registration

- Service providers have a two part problem:
  - Find lookup services
  - Register service proxies and maintain leases
- Service providers often use JoinManager
  - Keeps track of registrars, registrations, lease renewals, *etc.*
JoinManager

- `new JoinManager(Object, Entry [], ServiceID, DiscoveryManagement, LeaseRenewalManager)`
- `new JoinManager(Object, Entry [], ServiceIDListener, DiscoveryManagement, LeaseRenewalManager)`

- Provides accessor and mutators for changing attributes
Lookup

- Registrar’s lookup methods:
  - Takes:
    - ServiceTemplate
    - optional int count
  - Returns either:
    - Object (first matching service)
    - ServiceMatches (many matching services)
- ServiceMatches contain ServiceItems
  - Thereby making attributes available
ServiceTemplate

- ServiceID
- Class []
- Entry []
- Entry instances are:
  - Serializable
  - Zero arg constructor
  - Fields must be: public, serializable, non-transient, non-final, reference-types
Client Service Lookup

- Client’s usually have a multi-part problem:
  - Find appropriate lookup services
  - Find interesting services in those lookups
  - Track state/availability changes in services

- Client problem addressed by ServiceDiscoveryManager
  - Uses LookupDiscoveryManager, LookupLocatorDiscovery, or LookupDiscovery to find registrars
ServiceDiscoveryManager

- Constructor takes:
  - DiscoveryManagement
  - LeaseRenewalManager

- Can cache services matching a template
  - createLookupCache

- Lookup methods take:
  - ServiceTemplate
  - ServiceItemFilter (optional)
Example Code

- On CSS conference CD-ROM
Making It Work

- Jini isn’t hard, but many pieces must be right
- RMI class downloading requires:
  - **Accurate** `java.rmi.server.codebase` values
    - Consider setting in program code
    - Generally avoid using local classes
  - **Installed SecurityManager**
    - With sufficient permissions
- `-Dsun.rmi.loader.logLevel=VERBOSE`
- **Serializable objects**